Solution 1: Multiclass and Softmax Regression

(a) Read in the MNIST data set

```
library(keras)

## Error in library(keras):  there is no package called 'keras'

mnist <- dataset_mnist()

## Error in dataset_mnist():  could not find function "dataset_mnist"
```

(b) Visualize the data like

```
library(keras)

## Error in library(keras):  there is no package called 'keras'

mnist <- dataset_mnist()

## Error in dataset_mnist():  could not find function "dataset_mnist"

x_train <- mnist$train$x

## Error in eval(expr, envir, enclos):  object 'mnist' not found

y_train <- mnist$train$y

## Error in eval(expr, envir, enclos):  object 'mnist' not found

x_test <- mnist$test$x

## Error in eval(expr, envir, enclos):  object 'mnist' not found

y_test <- mnist$test$y

## Error in eval(expr, envir, enclos):  object 'mnist' not found

# visualize the digits
par(mfcol=c(1,6))
par(mar=c(0, 0, 3, 0), xaxs='i', yaxs='i')
for (idx in sample(1:NROW(x_train), 6)) {
    im <- x_train[idx,,]
    im <- t(apply(im, 2, rev))
    image(1:28, 1:28, im, col=gray((0:255)/255),
          xaxt='n', main=paste(y_train[idx]),
          yaxt='n')
}

## Error in NROW(x_train):  object 'x_train' not found
```

(c) Convert the features to a (`pandas`) data frame, by flattening the 28x28 images to a 784-entry-long vector, which represents one row in your data frame. Divide the intensity values of each pixel (each column) by 255 to get a value between 0 and 1.

```
library(tibble)
# reshape
dim(x_train) <- c(nrow(x_train), 784)

## Error in nrow(x_train): object 'x_train' not found

dim(x_test) <- c(nrow(x_test), 784)

## Error in nrow(x_test): object 'x_test' not found

# rescale
x_train <- x_train / 255

## Error in eval(expr, envir, enclos): object 'x_train' not found

x_test <- x_test / 255

## Error in eval(expr, envir, enclos): object 'x_test' not found

# convert to data.frame
x_train <- as_tibble(as.data.frame(x_train))

## Error in as.data.frame(x_train): object 'x_train' not found

x_test <- as_tibble(as.data.frame(x_test))

## Error in as.data.frame(x_test): object 'x_test' not found
```

(d) Softmax regression

```
library(nnet)
data <- cbind(y = as.factor(y_train), x_train)
# note: takes some time and requires quite some memory
# also you need to set the maximum number of weights to get it running
# we will further restrict the maximum number of iterations
# to avoid overfitting (explanation is given later)
model <- multinom(y ~ -1 + ., data = data, MaxNWts = 7860, maxit = 20)
```

```
## Error in is.factor(x): object 'y_train' not found
## Error in model.frame.default(formula = y ~ -1 + ., data = data): 'data' must be a
data.frame, environment, or list
```

Look at the larger weights:

```
summary(model$wts)

## Error in summary(model$wts): object 'model' not found

which.max(abs(model$wts))

## Error in which.max(abs(model$wts)): object 'model' not found

dim(coef(model))

## Error in coef(model): object 'model' not found
```

There seem to be a few very large coefficients

(e) Use `keras`:

```
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(keras)

## Error in library(keras):  there is no package called 'keras'

# convert outcome using one-hot encoding
y_train_one_hot <- to_categorical(y_train)

## Error in to_categorical(y_train):  could not find function "to_categorical"

y_test_one_hot <- to_categorical(y_test)

## Error in to_categorical(y_test):  could not find function "to_categorical"

neural_network <- keras_model_sequential()

## Error in keras_model_sequential():  could not find function "keras_model_sequential"

neural_network %>%
  layer_dense(units = 10, # corresponding to the number of classes
              activation = "softmax",
              input_shape = list(784)) %>%
  compile(
    optimizer = "adam",
    loss      = "categorical_crossentropy",
    metric = "accuracy"
  )

## Error in compile(., optimizer = "adam", loss = "categorical_crossentropy", :  could not
find function "compile"

history_minibatches <- fit(
  object           = neural_network,
  x                = as.matrix(x_train),
  y                = y_train_one_hot,
  batch_size       = 24,
  epochs           = 80,
  validation_split = 0.2,
  callbacks = list(callback_early_stopping(patience = 10)),
  verbose = FALSE, # set this to TRUE to get console output
  view_metrics = FALSE # set this to TRUE to get a dynamic graphic output in RStudio
)

## Error in fit(object = neural_network, x = as.matrix(x_train), y = y_train_one_hot, :
could not find function "fit"
```

Look at the network weights

```r
library(tensorflow)

## Error in library(tensorflow):  there is no package called 'tensorflow'

tensor_weights <- as.matrix(tf$add(neural_network$weights[[1]],0))

## Error in as.matrix(tf$add(neural_network$weights[[1]], 0)):  object 'tf' not found

summary(c(tensor_weights))

## Error in summary(c(tensor_weights)):  object 'tensor_weights' not found
```

and compare to the ones from multinomial logistic regression:

```r
plot(c(tensor_weights[,-1]) ~ c(t(coef(model))),
     xlab = "Weights Softmax Regression", ylab = "Weights Neural Network")

## Error in eval(predvars, data, env):  object 'tensor_weights' not found

abline(0,1, col="red")

## Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...):  plot.new has not
been called yet
```

As both models de facto are based on neural networks (here the implementation of the softmax regression is actually done by fitting a neural network with the very same network structure), their similarity depends on how the network is trained. While clearly the implementation calling Python with backend `TensorFlow` (the `keras` fit) is much much faster, the network also converges more quickly due to a small batch size while the multinomial logistic regression calls a network fitting algorithm that uses batch size equal to the number of observations (which is usually a bad idea).

(f) First define the metrics

```r
# Classification error (how many of the predictions are wrong)
classiferror <- function(actual, predicted) {
    return(mean(actual != predicted))
}

# Accuracy (how many of the predictions are correct)
accuracy <- function(actual, predicted) {
    return(1 - classiferror(actual, predicted))
}

# As we will usually have probabilistic predictions,
# we need to convert those to classes for the above
# metrics using the class with the max probability
probs_to_class <- function(probvec) {
  which.max(probvec)-1
}

#' MC Brier score
mcbrier <- function(actual_one_hot, prob) {
  rowSums((actual_one_hot-prob)^2)
}

# Cross-Entropy loss
```

```r
crossentropy <- function(actual_one_hot, prob) {
  rowSums( -log(prob) * actual_one_hot )
}

# negative log-likelihood of multinomial distribution
loglikmultinom <- function(actual_one_hot, prob) {
  sapply(1:nrow(actual_one_hot), function(i)
    dmultinom(actual_one_hot[i,],
              size = 1, prob[i,], log = TRUE))
}
```

Now we get the predictions:

```r
pred_multinom <- predict(model, x_test, type = "probs")

## Error in predict(model, x_test, type = "probs"):  object 'model' not found

pred_nn <- predict(neural_network, as.matrix(x_test))

## Error in predict(neural_network, as.matrix(x_test)):  object 'neural_network' not found

str(pred_multinom, 1)

## Error in str(pred_multinom, 1):  object 'pred_multinom' not found

str(pred_nn, 1)

## Error in str(pred_nn, 1):  object 'pred_nn' not found
```

Let's first look at the confusion matrix (in this case for the multinomial regression):

```r
table(y_test, apply(pred_multinom,1,probs_to_class))

## Error in table(y_test, apply(pred_multinom, 1, probs_to_class)):  object 'y_test' not
found
```

Now the metrics. Classification error:

```r
cbind(multinom = classiferror(y_test, apply(pred_multinom,1,probs_to_class)),
      neural = classiferror(y_test, apply(pred_nn,1,probs_to_class))
)

## Error in mean(actual != predicted):  object 'y_test' not found
```

Accuracy:

```r
cbind(multinom = accuracy(y_test, apply(pred_multinom,1,probs_to_class)),
      neural = accuracy(y_test, apply(pred_nn,1,probs_to_class))
)

## Error in mean(actual != predicted):  object 'y_test' not found
```

MC Brier score (note that we look at the mean, because the definition of the loss is on an observation basis):

```r
cbind(multinom = mean(mcbrier(y_test_one_hot, pred_multinom)),
      neural = mean(mcbrier(y_test_one_hot, pred_nn))
)

## Error in is.data.frame(x):  object 'y_test_one_hot' not found
```

Cross-entropy (mean):

```
cbind(multinom = mean(crossentropy(y_test_one_hot, pred_multinom)),
      neural = mean(crossentropy(y_test_one_hot, pred_nn))
)

## Error in is.data.frame(x):  object 'pred_multinom' not found
```

Mean negative log-likelihood of multinomial distribution:

```
cbind(multinom = mean(loglikmultinom(y_test_one_hot, pred_multinom)),
      neural = mean(loglikmultinom(y_test_one_hot, pred_nn))
)

## Error in nrow(actual_one_hot):  object 'y_test_one_hot' not found
```